Uma abordagem de escalonamento adaptativo no ambiente Real-Time CORBA

Alexandre Cervieri, Rômulo Silva de Oliveira, Cláudio F. Resin Geyer
cervieri@inf.ufrgs.br, romulo@lcmi.ufsc.br, geyer@inf.ufrgs.br

Universidade Federal do Rio Grande do Sul
UFRGS
Instituto de Informática
Av. Bento Gonçalves, 9500 – Bloco IV
Porto Alegre, RS – Brasil

Departamento de Automação e Sistemas
DAS-CTC-UFSC
Caixa Postal 476
CEP 88040-900
Florianópolis, SC – Brasil

Resumo. *CORBA vem se tornando o middleware padrão no desenvolvimento de aplicações distribuídas, tornando-as independentes de plataforma e linguagem. Ele tem sido utilizado também em aplicações de tempo real através de sua extensão para tempo real, o RT-CORBA. Apesar desta extensão ter conseguido contornar vários dos problemas do CORBA no que se refere ao não-determinismo e falta de garantias temporais, ainda há muito estudo na área de mecanismos de escalonamento utilizados. Assim, este trabalho tem por objetivo apresentar uma proposta de escalonamento adaptativo no ambiente Real-Time CORBA.*
Palavras-chave: sistemas distribuídos de tempo real, escalonamento adaptativo, TAO.

Abstract: *CORBA is becoming the standard middleware in the development of distributed applications, which could become platform and language independent. It has been used in many real-time applications, through its extension for real-time systems, RT-CORBA. When it comes to CORBA's drawbacks such as no determinism and the lack of time and QoS guaranties, RT-CORBA has succeded in achieving satisfactory solutions. Besides, there is still a lot of research for new scheduling mecanisms. So, the aim of this work is to present an adaptive scheduling on Real-Time CORBA enviroment.*
Keywords: real-time distributed systems, adaptive scheduling, TAO.

1. INTRODUÇÃO

O avanço das redes de computadores impulsionou a busca e o desenvolvimento de meios para facilitar e acelerar o desenvolvimento de aplicações em sistemas distribuídos. A partir de uma iniciativa da OMG (*Object Management Group*) de unificar padrões do mercado surgiu o CORBA (*Common Object Request Broker Architecture*), um *middleware* que busca uma maior independência de plataforma e linguagem [1] [2].

Entretanto, suas deficiências em termos de não-determinismo, falta de mecanismos para definir e garantir requisitos temporais e de QoS ainda deixavam-no longe das aplicações de tempo real. Assim, buscando suprir estas deficiências, a OMG partiu para uma extensão do CORBA para tempo real denominada de RT-CORBA [3] [4].

O RT-CORBA está chegando a sua segunda versão, mas ainda é um padrão em desenvolvimento. Foram criados interfaces e mecanismos para permitir a definição e a execução de uma grande variedade de aplicações de tempo real. Porém, muitos dos mecanismos que procuram manter a previsibilidade das aplicações dependem de outros aspectos como suporte do sistema operacional para garantir as exigências temporais, sistema de comunicação mais previsível e mecanismos e políticas de escalonamento que garantam a

execução das tarefas no momento correto [5].

Nesse último aspecto é que residem muitas das questões em aberto existentes no RT-CORBA. A escolha da política de escalonamento correta a uma determinada classe de aplicações é um dos pontos essenciais para o desenvolvimento de um sistema de tempo real. Aplicações de *hard real time* não permitem perdas ou atrasos no que se refere a tempo, por conseguinte, exige uma política de escalonamento com garantias em tempo de projeto. Já aplicações de *soft real time* permitem uma abordagem em tempo de execução e mesmo através de políticas de melhor esforço, onde detalhes podem ser excluídos para a manutenção do tempo da tarefa, ou mesmo pequenos atrasos são permitidos [6].

O objetivo deste trabalho é, portanto, apresentar um modelo de escalonamento adaptativo no ambiente RT-CORBA. Foram utilizados mecanismos do próprio RT-CORBA padrão e outros específicos do ORB TAO, o qual foi utilizado para desenvolvimento das experiências de validação da proposta.

Este artigo está organizado de modo que na seção 2 é feita uma breve revisão dos padrões CORBA e RT-CORBA além de apresentar algumas características da implementação utilizada, o TAO. A seção 3 é uma pequena revisão da literatura sobre mecanismos de adaptação em sistemas de tempo real. Finalmente, na seção 4 é apresentado o mecanismo de adaptação proposto e na seção 5 os resultados das experiências.

2. RT – CORBA

Definido pela OMG, o CORBA é, atualmente, uma das arquiteturas para desenvolvimento de aplicações em sistemas distribuídos mais completas. Muitos dos benefícios trazidos pelo CORBA residem na utilização de uma linguagem independente (IDL – *Interface Definition Language*) para a definição da interface dos objetos. Com a existência de mapeamentos de IDL para várias outras linguagens utilizadas no mercado, pode-se incorporar nos sistemas distribuídos até mesmo aplicações legadas e outros softwares não escritos originalmente para estes sistemas.

A OMA (*Object Management Architecture*) é definida como um modelo de referência para a tecnologia de objetos. O desenvolvimento dessa arquitetura seguiu certos objetivos técnicos para que trouxesse benefícios ao gerenciamento de objetos, tais como: conformidade, transparência de distribuição, desempenho em operações remotas e locais, comportamento extensível e dinâmico, arquitetura com serviço de nomes, controle de acesso, controle de concorrência, entre outros. Buscando atender esses objetivos a OMG (*Object Management Group*) buscou na OMA descrever uma generalidade, um modelo mais abstrato, ou seja, de mais alto nível, o qual o CORBA concretiza. Pode-se dizer que, portanto, a OMA serve como um “tipo” e o CORBA uma “instância” desse tipo [7].

O ORB (*Object Request Broker*) é o *middleware* de comunicação que permite que os clientes façam requisições e recebam respostas de objetos servidores, os quais podem se encontrar local ou remotamente em relação ao cliente. Foram definidos mapeamentos de IDL para as diversas linguagens existentes de modo a permitir o acesso ao ORB, deixando a sua forma de implementação livre. Isso, além de garantir uma liberdade para os desenvolvedores, assegura que um programa especificado para um ORB poderá ser parcialmente portável para qualquer outro ORB que seja compatível com as definições CORBA. O *Object Adapter*, por sua vez, tem a função de controlar o ciclo de vida dos servidores, ou seja, ser o responsável pela sua criação, ativação, destruição [8].

Clientes e servidores ainda têm a sua disposição os serviços padrão do ORB e os vários serviços CORBA já definidos pela OMG. Podemos destacar, pela sua importância neste trabalho: (i) o serviço de nomes, cuja função é identificar um objeto pelo seu nome quando é apresentada uma referência para o objeto e vice-versa; (ii) serviço de eventos, o qual define

uma interface genérica para envio de mensagens entre múltiplas fontes e múltiplos receptores. O serviço de eventos possui um canal de eventos que permite a geração de eventos sem que consumidores e fornecedores tenham que se comunicar diretamente. São oferecidas duas formas de interação: (i) *push*, o fornecedor é ativo, enviando eventos para o canal; (ii) *pull*, o consumidor é ativo, tenta receber um evento do canal (de forma bloqueante ou não) [1].

Apesar das vantagens que traz para o desenvolvimento de aplicações distribuídas, o CORBA 2.x não se enquadra em todas as exigências das aplicações para alto desempenho e tempo real, principalmente pelas seguintes razões [9] [10]: ausência de interfaces para definir QoS (*Quality of Service*), ausência de garantias de QoS, ausência de características de programação para tempo real e ausência de otimizações para alto desempenho.

Assim, em virtude dessa falta de suporte do CORBA para aplicações de tempo real, a OMG criou em 1995, um grupo de trabalho com o objetivo de estender as especificações CORBA. Em outubro de 1998, cinco propostas apresentadas foram unidas em um único documento [9], o RT-CORBA, que atualmente encontra-se em desenvolvimento sua versão 2, a qual passará a integrar o CORBA 3.

O RT-CORBA identifica capacidades que podem ser verticalmente (da camada de rede para a camada de aplicação e vice-versa) e horizontalmente (fim-a-fim) integradas e gerenciadas por um ORB para garantir previsibilidade nas atividades entre clientes e servidores CORBA. Seguem abaixo algumas dessas capacidades:

- **Gerenciamento dos recursos de comunicação**: o RT-CORBA deve prover políticas e mecanismos para gerenciar a infra-estrutura de comunicação, desde a escolha de uma conexão até a exploração das características avançadas de QoS, como por exemplo, interface para *threads*, estoque de *threads*, *binding* explícito, entre outras;
- **Mecanismos de escalonamento do sistema operacional**: ORBs exploram os mecanismos do sistema operacional para escalonar as atividades no nível da aplicação. O RT-CORBA se preocupa, inicialmente, em sistemas de tempo real com prioridade fixa. Dessa forma, estes mecanismos correspondem a gerenciar as prioridades das *threads* escalonadas pelo sistema operacional;
- **ORB de tempo real**: um ORB de tempo real deve prover interfaces padronizadas para permitir às aplicações especificar suas exigências de recursos para o ORB, além de fornecer comunicação entre os clientes e servidores de forma transparente;
- **Serviços e aplicações de tempo real**: um ORB de tempo real também deve criar um ambiente eficiente, escalonável e previsível fim-a–fim para serviços e aplicações de alto nível.

Visto que a OMG não tem como função fornecer uma implementação dos padrões CORBA e RT-CORBA, existem alternativas a serem analisadas. Dentre as implementações CORBA, mais especificamente, RT-CORBA, destaca-se o TAO, "*The ACE ORB*". Pode-se considerar o TAO mais do que apenas um ORB de comunicação de tempo real, mas sim uma arquitetura de um sistema final para execução de aplicações de tempo real, tanto *hard* quanto *soft*. A arquitetura do TAO contém as seguintes características e elementos [11]: subsistema de E/S de tempo real, RT-ORB, protocolo GIOP de tempo real, RT-POA, compilador IDL e camada de apresentação otimizadas, otimizações no gerenciamento de memória minimizando cópia de dados e meios para especificação de QoS.

O TAO é um dos ORBs que apresenta a implementação mais completa do RT-CORBA. Entretanto, alguns serviços por ele implementados diferem das definições do padrão. Um desses serviços é o de escalonamento. O serviço de escalonamento do TAO é responsável por alocar os recursos do sistema a fim de garantir as necessidades das aplicações. Sendo mais direcionado para aplicações de *hard real time*, seu principal objetivo é garantir que as exigências de recursos sejam asseguradas. É dividido em componentes de tempo de projeto

(*offline*) e de tempo de execução (*runtime*). Na primeira, é analisada a possibilidade de escalonamento correto de todas as operações e são atribuídas as suas prioridades. Os componentes de tempo de execução oferecem acesso rápido aos valores de prioridades e coordena as mudanças de modos.

Outro serviço do TAO que foi melhorado para o uso em aplicações de tempo real foi o serviço de eventos. Os itens contemplados afetam basicamente o método de interação *push*, sendo que os principais são [12]: (i) consumidores e fornecedores de eventos podem especificar suas exigências e características de execução usando parâmetros de QoS; (ii) mecanismos de correlação e filtragem de eventos centralizados no canal de eventos; (iii) consumidores podem especificar eventos dependentes de *timeouts*.

3. MECANISMOS DE ADAPTAÇÃO EM APLICAÇÕES DE TEMPO REAL

Mesmo em sistemas computacionais construídos especificamente para a execução de aplicações de tempo real existem amplas oportunidades para adaptação. Isto pode ocorrer porque a carga representada pela aplicação em questão não possui um limite conhecido que possa ser garantido em projeto, ou porque, mesmo que a carga possua uma demanda por recursos bem caracterizada, pode não ser economicamente viável garantir o seu comportamento em um cenário de pior caso.

Embora a maior parte da literatura de tempo real trate de adaptação através da negociação da qualidade de serviço, a adaptação também pode ocorrer através de uma ação unilateral. No caso de uma adaptação unilateral existem os seguintes cenários possíveis [6]:

- Uma variação no comportamento da aplicação gera como resposta uma adaptação do suporte.
- Uma variação no comportamento do suporte gera como resposta uma adaptação do próprio suporte.
- Uma variação na aplicação gera como resposta uma adaptação da própria aplicação. Em sistemas onde a aplicação pode reservar recursos cabe a própria aplicação administrar os recursos reservados.
- Uma variação no comportamento do suporte gera como resposta uma adaptação da aplicação. Esta situação é muito comum em sistemas distribuídos, onde variações no atraso podem estar associadas com o envio de mensagens pela rede. A diferença entre os processadores que compõem a rede também pode ser causa de alterações nos atrasos.

Aplicações de tempo real em ambiente distribuído podem estar sujeitas a variações nos tempos de resposta das tarefas. Essas variações podem ser causadas por atrasos no envio de mensagens pela rede ou na alteração de algum nodo do qual a aplicação também é dependente. Isto pode acontecer ao longo do tempo de execução da aplicação, e não apenas em sua etapa de inicialização. Logo, existe a necessidade de uma adaptação contínua durante a execução da aplicação. Nesta seção serão examinados mecanismos de adaptação descritos na literatura e que podem ser empregados no contexto considerado:

- **Atrasar a conclusão da tarefa.** A forma mais simples e freqüente de adaptação é simplesmente relaxar o conceito de *deadline*. Um dos primeiros trabalhos seguindo esta abordagem aparece em Jensen et. al. [13], onde é proposto que a conclusão de cada tarefa contribui para o sistema com um benefício e o valor deste benefício pode ser expresso em função do instante de conclusão da tarefa (*time-value function*).
- **Variar o período da tarefa.** Em uma aplicação tempo real muitas tarefas são executadas periodicamente. Em geral estas tarefas possuem o seu período definido em tempo de projeto. Uma forma de prover adaptabilidade à aplicação é permitir que este período possa

variar dinamicamente, durante a execução da aplicação. Desta forma, a qualidade da aplicação, representada aqui pelo período das tarefas, seria adaptada ao desempenho do suporte onde estiver executando.

- **Cancelar a execução de uma tarefa.** Uma forma mais radical de flexibilização é simplesmente não executar algumas tarefas quando o desempenho estiver abaixo do desejado. No caso de aplicações com tarefas repetitivas, é possível cancelar uma ativação específica da tarefa ou cancelar completamente a tarefa.
- **Alterar o grau de paralelismo.** Muitos algoritmos dividem o trabalho a ser feito em partes. As partes podem então ser executadas em paralelo, com o objetivo de diminuir o tempo de resposta da tarefa. Caso a arquitetura sendo usada não suporte execução em paralelo, as partes são executadas seqüencialmente. O resultado é o mesmo, apenas o tempo de execução será maior.
- **Variar o tempo de execução da tarefa.** Neste mecanismo de adaptação, as tarefas são escalonadas de forma a cumprirem seus *deadlines*. Em caso de sobrecarga, o tempo de execução da tarefa é reduzido. Para isto, é necessário que cada tarefa possua opções do tipo "qualidade versus tempo de execução". Esta abordagem é denominada na literatura de "Computação Imprecisa".

Na literatura são encontradas várias propostas de como medir a qualidade do sistema e de como atuar no mesmo buscando o benefício dessa medida. Por exemplo, em Lu et. al. [14] a qualidade do sistema é medida através da taxa de perda de *deadline* das tarefas e da taxa de utilização do sistema em uma janela de medição. A atuação é então feita de modo a alterar o modo de operação das tarefas e na decisão de aceitar ou rejeitar tarefas para execução. Beccari et. al. [15] também identifica sobrecargas no sistema através da utilização do processador e a atuação é feita através de uma política de *graceful degradation* do período das tarefas. Outros trabalhos encontrados na literatura que seguem pela mesma direção podem ser encontrados em Welch et. al. [16], Shin & Meissner [17] e Abdelzaher et. al. [18] [19].

4. MECANISMO DE ADAPTAÇÃO PROPOSTO

4.1 Descrição

Na seção 3 foram apresentados alguns dos mecanismos de adaptação encontrados na literatura. Considerando que muitas aplicações de tempo real têm sua estrutura baseada em atividades periódicas, optou-se por uma abordagem mais direcionada a esse tipo de atividade. Ainda, a proposta deste trabalho foi inspirada em Lu et. al. [14] e outros trabalhos que utilizam realimentação para controlar o sistema em sobrecarga.

Da mesma forma que em Lu et. al. [14], a atuação no sistema é dependente das informações que são coletadas junto às atividades que compõem o sistema e analisadas por um processo periódico, o qual neste trabalho será denominado de *AdaptiveService*. Lu et. al. [14] analisa, para adaptação do sistema, as informações de taxa de utilização do sistema (U(t)) e a taxa de perda de *deadline* (MR(t) – *miss ratio function*) das tarefas em uma janela de medição (MW – *miss-ratio window*). A medida da taxa de perda de deadline é bastante adequada para sistemas com *deadline* firme (*firm-deadline*), ou seja, onde não existe benefício em executar uma tarefa após o seu *deadline*.

Ao contrário daquela abordagem, que foi criada para um sistema monoprocessado, estamos tratando de sistemas distribuídos e ainda, buscamos neste trabalho considerar tarefas não tão exigentes, onde mesmo perdendo seus prazos a execução das tarefas ainda é importante. Assim, a forma de quantificar o desempenho do sistema foi através da medida comparativa entre os *deadlines* das tarefas com os tempos de resposta efetivamente observados. Esta medida comparativa foi denominada de *delay profile* (DY(t)), o qual é observado após uma

janela de medição denominada de *delay window* (DW). Neste trabalho, o *deadline* de cada tarefa está sendo considerado como o período da mesma (D=P), portanto, DY(t) pode ser expresso por:

$$DY(t) = \sum_{t-DW}^{t} \frac{(R_i - D_i)}{N}$$

t: instante de tempo atual
DW: *delay window*, janela de tempo para cálculo de DY(t)
R_i: tempo de resposta da tarefa *i* para uma determinada ativação
D_i: *deadline* da tarefa
N: número de conclusões (de todas as tarefas) na janela DW

A atuação é feita sobre os períodos das tarefas que compõe o sistema. Esta atuação poderia ser feita de forma específica para cada tarefa. Possuindo um fator de adaptação calculado especialmente para cada tarefa diferenciando-as conforme sua prioridade, por exemplo. Considerando que todas as tarefas foram escalonadas (utilizando uma política como *Rate Monotonic*, por exemplo) o fator de diferenciação entre as tarefas já foi atribuído como sendo sua prioridade, caso a plataforma (sistema operacional e/ou *middleware*) respeite esta atribuição, uma atuação especifica a cada tarefa não seria necessária. Assim, DY(t) é utilizado para o cálculo de um *"fator"* de atuação sobre o período de cada tarefa da seguinte forma:

$$fator = K_P * E(t) \Rightarrow fator = K_P * (DY(t) - SDY) \Rightarrow fator = K_P DY(t) - K_P SDY$$

Sendo que E(t) é o valor do erro do sistema em um determinado instante, o qual pode ser expresso por DY(t)-SDY, onde SDY é o atraso médio desejável para o sistema. K_P (constante proporcional) e SDY são valores atribuídos pelo projetista e/ou programador do sistema segundo cálculos matemáticos ou experimentação. Quanto menor o valor de K_P mais suave será a atuação no sistema.

O *fator* é então utilizando pelo *AdaptiveService* pelo cálculo do período efetivo de cada tarefa conforme:

$$P_{efetivo} = P_{mínimo} + fator(P_{máximo} - P_{mínimo})$$

Além dos valores de K_P, SDY e DW, a forma de o projetista/programador especificar a atuação se estende para os valores de $P_{mínimo}$ (período mínimo) e $P_{máximo}$ (período máximo) de cada tarefa. Segundo a fórmula acima, o *AdaptiveService* é capaz de variar o período de cada tarefa (período efetivo) segundo o *fator* calculado anteriormente sempre dentro dos limites estabelecidos para o período mínimo e máximo de cada uma.

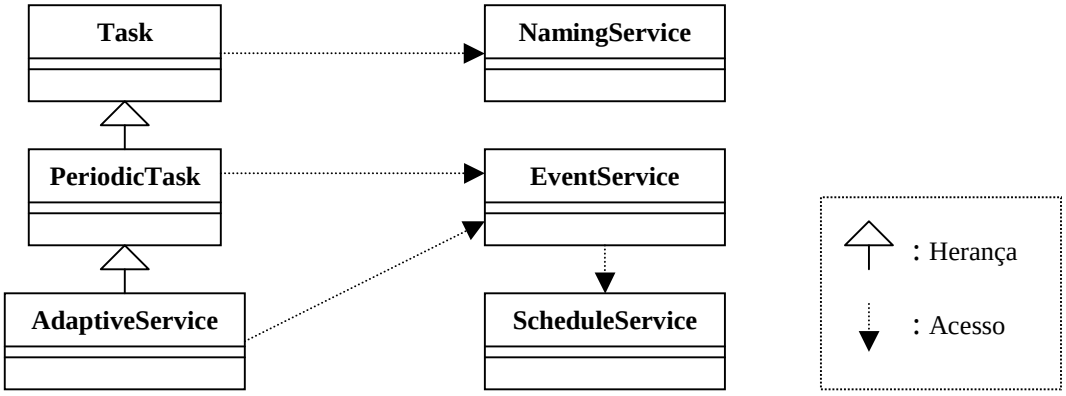


Figura 1 - Diagrama da implementação do modelo de adaptação.

4.2 Implementação do mecanismo de adaptação no contexto do ORB TAO

O mecanismo de adaptação proposto se destina a aplicações compostas por tarefas periódicas. Porém aplicações reais também podem conter tarefas não periódicas e passivas, assim o diagrama de classes da implementação do mecanismo de adaptação, o qual pode ser visto na

Figura 1, contempla também este tipo de tarefa.

Os principais elementos da Figura 1 são detalhados nos próximos itens desta seção.

Task

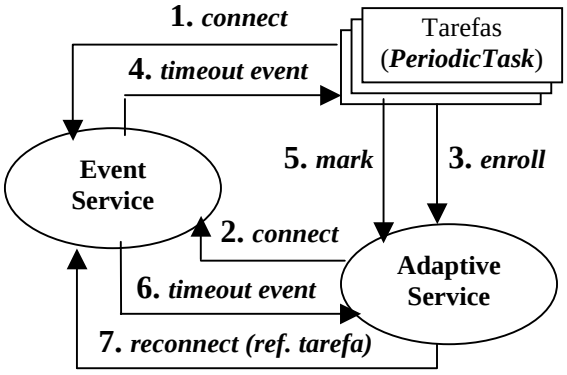
Representa as tarefas não periódicas e passivas. É implementado como um *servant* que irá encarnar um objeto CORBA. No momento de sua criação registram um nome no serviço de nomes (***NamingService***) para que os objetos ativos do sistema possam encontrá-los e realizar invocações de métodos.

Cada tarefa, objeto CORBA, é ativada em um POA próprio, configurado com a política de *ClientPropagatedPriority*, de forma a respeitar a prioridade dos objetos que invocam seus métodos.

PeriodicTask

Representa as tarefas periódicas. Assim como *Task*, essa classe é abstrata, a qual não pode ser diretamente instanciada, necessita da definição de uma classe derivada. Uma tarefa periódica (*PeriodicTask*) é, em última instância, um consumidor que se registra no serviço de eventos (*EventService)* do TAO para receber eventos de *timeout* em intervalos específicos de tempo, este intervalo é o período definido na *RT_Info* da tarefa na fase *offline* do escalonamento.

Ainda, cabe ressaltar que cada *PeriodicTask* também possui um POA próprio, configurado com a política de *ServerDeclaredPriority* (de forma a utilizar a prioridade definida em sua *RT_Info*) e um *ThreadPool* com número de *threads* estáticas e dinâmicas que pode ser alterado pelo programador no momento de criação de instâncias.



1. Cada tarefa se registra no serviço de eventos para receber eventos de *timeout* em intervalos definidos pelo período cadastrado na *RT_Info* na fase *offline* da aplicação.
2. O serviço de adaptação também se cadastra para eventos de *timeout* informando o período no serviço de eventos.
3. Cada tarefa cadastra suas informações de períodos máximo e mínimo (enviando também sua referência) no serviço de adaptação.
4. Depois das tarefas se conectarem, o serviço de eventos passa a gerar eventos de *timeout* em intervalos definidos pelo período da tarefa cadastro na sua *RT_Info*.
5. Após cada ativação de uma tarefa, é medido e enviado ao serviço de adaptação o tempo de resposta para aquela ativação.
6. Em cada ativação, o serviço de adaptação calcula o valor de DY(t) para a janela DW e o novo valor do fator.
7. Caso o fator da ativação atual seja diferente da ativação anterior o *AdptiveService* solicita ao serviço de eventos a reconexão de cada uma das tarefas informando o novo período efetivo.

Figura 2 - Coleta de informações e atuação do Serviço de Adaptação.

AdaptiveService

O serviço de adaptação (*AdaptiveService*) também foi implementado como uma tarefa periódica que precisa ser cadastrada como uma *RT_Info* na fase *offline* do escalonamento. Seu período, o qual assume-se igual a DW, pode ser configurado na *RT_Info*.

O serviço de adaptação é o responsável pela coleta dos dados junto às tarefas periódicas (as tarefas não periódicas não são contabilizadas) e posterior atuação no sistema. Este processo

pode ser visto na Figura 2.

Para utilizar o serviço de escalonamento do TAO é necessário que a aplicação seja dividida em duas fases: *offline* e *runtime*. Na fase *offline* é feito o cadastramento das características temporais de cada uma das tarefas (e no caso do serviço de adaptação também) em estruturas chamadas de *RT_Infos*.

No início da fase de *runtime* cada tarefa periódica deve obter uma referência para o serviço de adaptação e invocar o método *enroll* para se cadastrar junto ao mesmo, informando os valores de período mínimo e máximo admitido. Ainda, cada tarefa periódica deve ser implementada como uma instância de uma classe derivada de *PeriodicTask*. Isto porque esta classe provê mecanismos que obtém o tempo de resposta da tarefa através de *timers* e automaticamente invocam o método *mark* do serviço de adaptação para informar o valor medido.

O serviço de adaptação tem, portanto, a função de a cada ativação (em intervalos de DW) calcular o valor de DY(t) e o valor do *fator* de adaptação dos períodos das tarefas. O $P_{efetivo}$ calculado para cada tarefa é então utilizado para solicitar, junto ao serviço de eventos, a reconexão de cada tarefa para eventos de *timeout*, mas com este novo valor como intervalo entre ativações. O tempo de resposta do serviço de adaptação a aumentos de carga é, portanto, dependente do tempo de resposta do serviço de eventos para re-conectar os consumidores para novos intervalos de eventos de *timeout*. Situações que exijam respostas imediatas a aumentos repentinos de carga necessitariam de um serviço de eventos com maior rapidez no que se refere a reconexões de consumidores.

5. EXPERIÊNCIAS

5.1 Estrutura da aplicação experimental

Pode-se encontrar inúmeras classes de aplicações de tempo real em ambiente distribuído que poderiam ser utilizadas como objeto de teste do mecanismo de adaptação proposto neste trabalho.

Um tipo de aplicação de tempo real que possui muitas possibilidades de estudo é a dos sistemas de automação industrial. Neste tipo de sistema podem ser encontradas tarefas periódicas, aperiódicas, esporádicas, tarefas ativas e reativas e tarefas com diferentes prioridades dentro do sistema.

Procurou-se neste trabalho identificar e representar alguns dos tipos de tarefas encontradas neste tipo de sistemas.

- **Sensores** e **Atuadores** foram os elementos da classe de aplicações de sistemas de automação escolhidos para representar as tarefas passivas. São derivados da classe *Task* e possuem métodos definidos em IDL que podem ser invocados por elementos ativos do sistema. Seu comportamento interno simula o tempo de processamento que é cadastrado junto ao serviço de escalonamento na fase *offline* do escalonamento.
- Nos sistemas de automação foram identificadas três possíveis classes de tarefas periódicas, que são definidas como classes derivadas de *PeriodicTask*: (i) **Historico** representa as tarefas que são responsáveis pela coleta de dados de *log*, de modo a armazenar a situação do sistema ao longo do tempo, considerada uma tarefa de pouca prioridade e com um período razoavelmente alto (como um segundo); (ii) **Operador** representa aqueles processos de monitoramento (realizando coleta e apresentação das informações) que tem interação com o operador do sistema; (iii) a classe **Alarme** representa processos de monitoramento de informações que, em determinados casos poderiam identificar problemas no sistema, supõem-se, então, que seu período seja menor que das outras tarefas a fim de estar sempre atualizados com os últimos dados coletados no sistema. Essas três

tarefas também simulam um tempo de processamento que é informado na *RT_Info*.

O comportamento de cada classe definida acima, além de simular o tempo de execução determinado na fase *offline* da aplicação também reflete as dependências[1], seja *oneway* ou *twoway* definidas. Os detalhes sobre o cadastro destas tarefas, suas restrições temporais e dependências pode ser visto na próxima seção deste trabalho.

5.2 Cenário das experiências

De modo a validar o mecanismo de adaptação proposto, foram realizados conjuntos de experimentos considerando um mesmo cenário de teste. Considerando a classe de aplicações escolhida como base de teste para o trabalho (automação industrial), onde em muitos casos a configuração inicial do sistema nunca é alterada, mantendo-se o mesmo número de tarefas e o mesmo número de equipamentos de *hardware* (sejam computadores, sensores, atuadores, etc.) ao longo do tempo, procurou-se, da mesma forma, manter o mesmo cenário de teste. Realizando alterações apenas no mecanismo de adaptação proposto.

Assim, para todos os experimentos mostrados nesta seção, foi utilizada uma rede formada por duas máquinas, com configuração de *hardware* e *software* apresentada na Tabela 1.

Tabela 1 - Configuração de *hardware* e *software* do cenário das experiências

Máquina	***Hardware***	***Software***
Nodo 1	- Processador *Pentium* III 650MHz - 128MB de memória RAM	- Sistema Operacional *Windows* 2000 *Advanced Server* - Compilador *Visual* C++ 6.0 - ACE versão 5.2 - TAO versão 1.2
Nodo 2	- Processador AMD K7 *Duron* 750MHz - 128MB de memória RAM	- Sistema Operacional *Windows* 2000 *Advanced Server* - Compilador *Visual* C++ 6.0 - ACE versão 5.2 - TAO versão 1.2

As tarefas foram distribuídas entre as máquinas de forma arbitrária e também estática. São inúmeras as possibilidades de atribuição das tarefas às máquinas, assim como são inúmeros os argumentos para a escolha de uma ou outra forma. Poderia-se utilizar uma atribuição de tarefas de forma que a carga se mantivesse homogeneamente distribuída entre as máquinas. Ainda, poder-se-ia utilizar critérios de afinidade entre as tarefas, como tarefas com uma grande comunicação entre si permanecerem juntas em uma mesma máquina. Entretanto, para uma aplicação de teste todo o estudo necessário para a correta distribuição das tarefas pode ser deixado de lado em um primeiro momento, o que certamente não pode ser feito no caso de uma aplicação real.

Assim, a Tabela 2 mostra a alocação de tarefas utilizada nas experiências, bem como seus tempos de execução, período, dependências cadastradas no serviço de escalonamento e a prioridade calculada por este.

[1] O tipo da dependência é referente à forma de invocação de método de um objeto. Dependência do tipo *oneway* se refere a invocações a métodos definidos com essa palavra chave em IDL, significando que o método não bloqueia o objeto chamador. No caso de dependência *twoway*, o método bloqueia o objeto chamador até o recebimento de uma resposta.

Tabela 2 - Alocação de tarefas entre os nodos da rede.

Máquina	Tarefas						
	Nome	Pior Tempo de Execução	Período	Dependências		Prioridade	
				oneway	*twoway*	CORBA[2]	Win 2000
Nodo 1	sensorA	10 ms	-	-	-	2	1
	atuadorA	20 ms	-	-	-	0	15
	atuadorB	20 ms	-	-	-	0	15
	alarme1	20 ms	100 ms	atuadorA atuadorB	sensorA sensorB	0	15
	operador1	300 ms	500 ms	-	sensorA sensorB	1	2
Nodo 2	sensorB	10 ms	-	-	-	2	1
	alarme2	20 ms	100 ms	atuadorA atuadorB	sensorA sensorB	0	15
	operador2	300 ms	500 ms	-	sensorA sensorB	1	2
	historico1	500 ms	1000 ms	-	sensorA sensorB	2	1
	historico2	500 ms	1000 ms	-	sensorA sensorB	2	1
	AdaptiveService	40 ms	1000 ms	-	-	2	1
	NamingService	-	-	-	-	-	-
	EventService	-	-	-	-	-	-
	ScheduleService	-	-	-	-	-	-

5.3 Resultados obtidos

Inicialmente foram feitas execuções do sistema sem que o serviço de adaptação (*AdaptiveService*) atuasse de qualquer forma no sistema, mas apenas calculasse o DY(t) a cada período de sua execução. Assim, a Tabela 3 mostra as médias dos valores encontrados de período e atraso das tarefas periódicas e a Figura 3 mostra a evolução do DY(t) ao longo do tempo (para execuções de dois minutos).

Tabela 3 - Média dos valores de período e atraso do sistema sem adaptação.

Tarefa	Média do período (P)	Média do atraso (R-D)[3]
alarme1	102,90 ms	541,76 ms
alarme2	104,59 ms	169,02 ms
operador1	499,86 ms	-236,42 ms
operador2	500,08 ms	-275,51 ms
historico1	1000,24 ms	-776,13 ms
historico2	999,77 ms	-758,42 ms

Os valores negativos para o atraso na Tabela 3 indicam sobra de tempo.

[2] O valor zero representa maior prioridade e quanto maior o valor, menor é a prioridade da tarefa.
[3] R é o tempo de resposta da tarefa para uma dada ativação e D é o deadline da tarefa. Neste trabalho está sendo considerado o deadline como sendo igual ao período da tarefa (D=P).

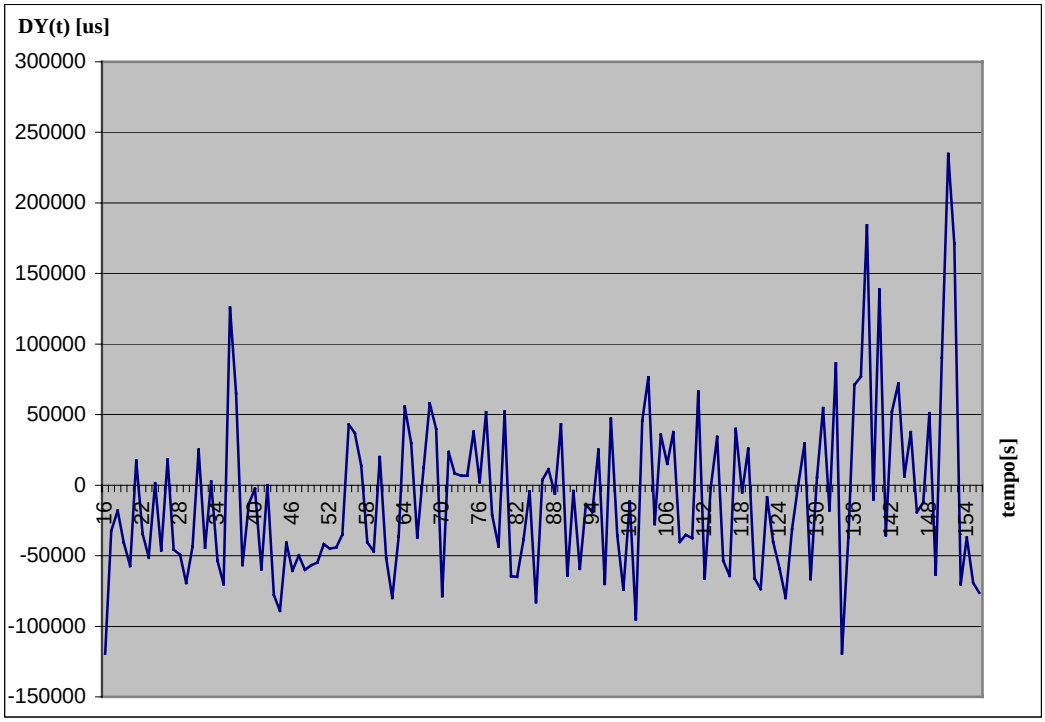


Figura 3 - Evolução do DY(t) ao longo do tempo para o sistema sem adaptação.

Pela análise da Tabela 3 e Figura 3 se observa que ocorrem atrasos significativos afetando com mais rigor as duas tarefas de maior prioridade (alame1 e alarme2). Apesar de possuir a maior prioridade, esses processos são também os mais exigentes em termos de ocupação de processador, acabam sofrendo atrasos por este motivo, pela própria característica do *Windows* de não ser um sistema operacional preemptivo por prioridade puro e também pela característica do protocolo TCP/IP de apresentar filas de envio baseado na política de FIFO.

Em um primeiro momento foi implementada uma versão ainda grosseira de um algoritmo de adaptação. Neste algoritmo, em cada ativação do serviço de adaptação (foi utilizado o período de um segundo) é calculado o valor atual de DY(t) e então comparado com o valor de SDY (valor que se assumiu como sendo zero). Caso DY(t) fosse maior que SDY então o serviço de adaptação atua no sistema incrementando em um o fator de multiplicação dos períodos das tarefas e então atualizando no serviço de eventos o período de cada tarefa cadastrada no serviço de adaptação. Caso DY(t) seja menor que SDY por dois ou três ciclos consecutivos (valor configurado pelo programador), o fator de multiplicação dos períodos das tarefas é decrementado de um (contanto que seja sempre maior ou igual a um) e então os períodos são atualizados no serviço de eventos. Os resultados encontrados utilizando-se esta primeira versão de adaptação podem ser vistos na Tabela 4 e Figura 4.

Tabela 4 - Média dos valores de período e atraso do sistema com a primeira forma de adaptação.

Tarefa	Média do período (P)	Média do atraso (R-D)
alarme1	157,32 ms	41,25 ms
alarme2	157,66 ms	40,93 ms
operador1	887,16 ms	-217,10 ms
operador2	950,07 ms	-360,42 ms
historico1	1908,04 ms	-697,72 ms
historico2	1883,43 ms	-608,05 ms

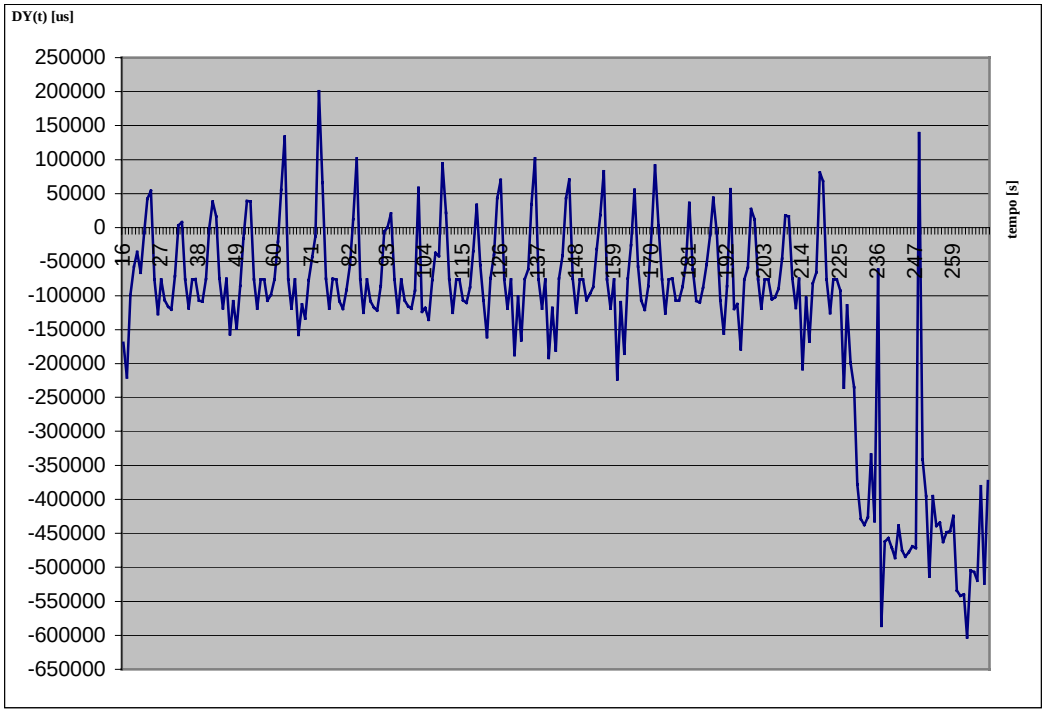


Figura 4 - Evolução do DY(t) ao longo do tempo com a primeira forma de adaptação.

A partir do instante que algumas tarefas param de executar, o sistema passa a apresentar tempos de atraso menores, momento no qual o serviço de adaptação passa a não atuar, ou atuar menos no sistema. Isto ocorre, na Figura 4, por volta do instante 230. Portanto, nos demais gráficos apresentados, este momento de término das tarefas não será mais mostrado.

Com esta primeira versão do mecanismo de adaptação, o sistema apresentou uma melhora sensível nos tempos de média de atraso. Entretanto, pela visualização do gráfico de evolução de DY(t) pode-se notar que a adaptação ainda insere instabilidade no sistema. Isto se deve a sua forma de atuação ser ainda excessivamente forte mesmo com pequenos índices de atraso. Assim, buscando-se um ajuste mais fino da adaptação, foi utilizado o mecanismo de adaptação descrito na seção 4.1 no restante das experiências.

Tabela 5 - Médias dos tempos na primeira rodada com a segunda versão de adaptação.

Tarefa	Média do período (P)	Média do atraso (R-D)
alarme1	114,17 ms	34,54 ms
alarme2	115,52 ms	11,10 ms
operador1	666,49 ms	-337,85 ms
operador2	660,45 ms	-305,08 ms
historico1	1566,89 ms	-696,92 ms
historico2	1558,97 ms	-658,29 ms

Um dos primeiros testes realizados foi utilizando-se K_P=1/250000 e SDY=-50000 us. Os resultados podem ser vistos na Tabela 5 e Figura 5. Observa-se que os tempos de atraso são um pouco menores (mas ainda existentes) que aqueles encontrados com a primeira versão da adaptação, mas que a adaptação continua um pouco instável, com oscilações em torno do valor determinado para SDY. Além disso, nota-se que os atrasos diminuíram mesmo com as tarefas (como alarme1 e alarme2) executando com períodos médios menores (de aproximadamente 157ms, na primeira versão, para aproximadamente 115ms nas tarefas alarme1 e alarme2).

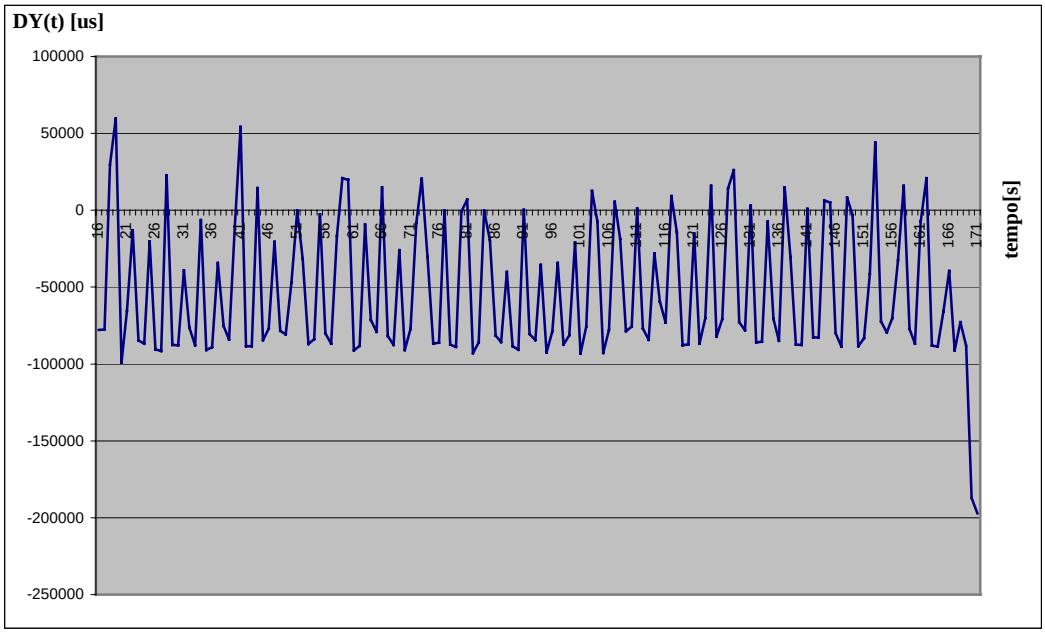


Figura 5 - Evolução de DY(t) na primeira rodada com a segunda versão de adaptação.

A estabilidade do controle feito pelo mecanismo de adaptação depende dos valores associados à K_P e SDY. O ajuste destas constantes poderia ser feito através de cálculos matemáticos da área de engenharia de controle, relativos à política de controle proporcional. Porém, não era objetivo deste trabalho aprofundar o estudo nesta área. Desta forma, procurou-se, através de experimentação, alterar os valores das constantes K_P e SDY de modo que a atuação no sistema se tornasse mais suave e ainda que, no caso da existência de pequenas oscilações, elas se mantivessem em valores negativos de DY(t).

Após estes testes chegou-se aos valores de K_P=1/500000 e SDY=-100000 us, cujos resultados podem ser vistos na Tabela 6 e Figura 6.

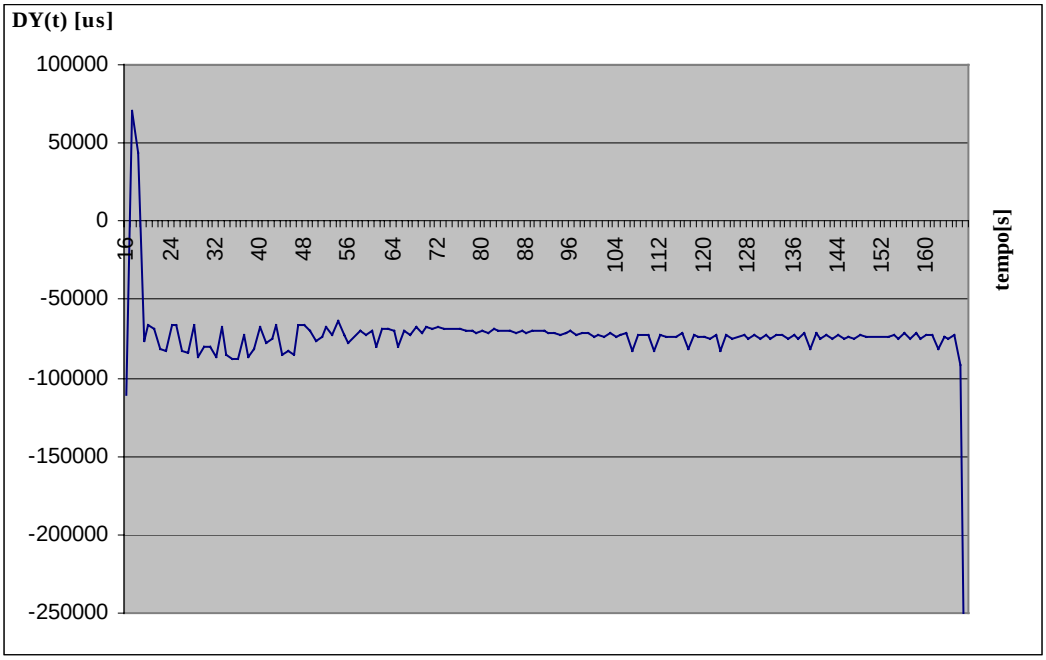


Figura 6 - Evolução de DY(t) para o sistema com adaptação melhor configurada.

Tabela 6 - Médias dos tempos para o sistema com adaptação melhor configurada.

Tarefa	Média do período (P)	Média do atraso (R-D)
alarme1	111,13 ms	-55,70 ms
alarme2	111,15 ms	-35,04 ms
operador1	762,33 ms	-364,20 ms
operador2	760,32 ms	-346,55 ms
historico1	1000,11 ms	-709,02 ms
historico2	1000,05 ms	-697,02 ms

Observa-se na Figura 6 a maior estabilidade da atuação sobre o sistema. Após o pico inicial de carga o sistema atua de forma a melhorar o DY(t) do sistema. O deslocamento de SDY para um valor mais negativo (no caso, -100000) teve, como conseqüência, a manutenção do valor de DY(t) próximo deste valor devido a atuação do serviço de adaptação. Então, as médias dos atrasos (observados na Tabela 6) se mantiveram negativas para todas as tarefas, significando que, com pequenas alterações nos períodos (os quais ficaram, em sua maioria, até abaixo dos períodos médios resultantes mostrados na Tabela 5) das tarefas pelo serviço de adaptação, todas conseguiram manter pequenas folgas em relação a seus *deadlines*.

Os valores das constantes K_P e SDY vão depender das condições de cada aplicação. Como afirmado anteriormente, é necessário que o engenheiro de controle ou o projetista do sistema determine os valores mais adequados para esses parâmetros conforme cálculos matemáticos ou medições experimentais feitos sobre a sua aplicação específica.

Os resultados apresentados nesta seção mostram que, mesmo lidando com um conjunto de tarefas apenas representativas, o mecanismo de adaptação proposto pode ser utilizado em casos reais com as referidas configurações necessárias para cada aplicação.

6. CONCLUSÕES E TRABALHOS FUTUROS

O CORBA trouxe vantagens para o desenvolvimento de aplicações distribuídas. Porém, apresenta deficiências na utilização em aplicações de tempo real que impulsionaram a definição de uma extensão específica para este fim, o RT-CORBA. O RT-CORBA é ainda um padrão em desenvolvimento. As implementações existentes tentam seguir suas recomendações, mas em certos momentos acabam por criar soluções próprias para tornar seus ORBs mais determinísticos e com características a mais para a criação de aplicações de tempo real. Mesmo considerando implementações como o TAO, uma das mais completas de RT-CORBA encontradas e que ainda acrescenta melhorias, alguns pontos passam em branco.

O aspecto mais importante a ser levado em consideração, é que o CORBA e sua extensão RT-CORBA ainda é um *middleware*, e por este motivo depende muito do sistema operacional e do *hardware* sobre o qual é colocado a executar. Por este motivo, por mais que o RT-CORBA (ou, mais especificamente, uma implementação como o TAO) se aproxime e solucione suas deficiências de definição e garantia de tempo e QoS, estruturas e forma de programação para tempo real, ele ainda é dependente e precisa que tanto o sistema operacional, *hardware*, quanto a estrutura de rede, suporte as exigências de tempo e qualidade de serviço que as aplicações de tempo real necessitam.

Pensando nesses aspectos que podem afetar a qualidade de uma aplicação tempo real, e se preocupando especialmente com aplicações que suportassem uma abordagem de melhor esforço, foi apresentado neste artigo um mecanismo de adaptação para o ambiente RT-CORBA.

O foco principal de preocupação foi com as tarefas periódicas que possuem restrições temporais. Embora a abordagem apresentada não seja possível em absolutamente todas as aplicações, ela pode ser usada sempre que existir alguma flexibilidade no período das tarefas.

A atuação é baseada em um laço de realimentação que fornece as informações de tempo de resposta em uma ativação de cada tarefa para um serviço de adaptação. Escolheu-se como alvo da adaptação o período das tarefas. A variação do período é derivada do cálculo do que se chamou de DY(t), *delay profile*, o qual é, em última instância, a medida de quanto as tarefas perderam ou ganharam em relação ao seu *deadline* (período) em uma janela específica de medição (DW, *delay window*).

A idéia do mecanismo apresentado neste artigo é basicamente sacrificar o período das tarefas para manter sob controle os atrasos, ao mesmo tempo em que exerce controle sobre o comportamento do sistema em sobrecarga. Ao contrário da maioria dos sistemas, onde a sobrecarga gera uma degradação aleatória, a abordagem proposta permite uma degradação controlada nos momentos de sobrecarga. É também importante notar que a adaptação é dinâmica, capaz de responder às flutuações de carga que ocorram durante a execução do sistema. O mecanismo automaticamente obtém do sistema a maior qualidade possível, porém respeitando os *deadlines* das tarefas.

A validação do mecanismo proposto foi feita através da sua implementação utilizando os mecanismos do RT-CORBA e os serviços de escalonamento e eventos que aparecem, de forma diferenciada, no ORB TAO. Os resultados mostram que para uma dada aplicação experimental o mecanismo conseguiu melhorar a qualidade do sistema representada pela comparação entre o tempo de resposta e o *deadline* das tarefas.

O mecanismo de adaptação utilizado atua de forma homogênea sobre as tarefas do sistema, é utilizado o mesmo *fator* de adaptação ao período das tarefas. Melhorias poderiam ser obtidas em sistemas operacionais como o *Windows*, o qual não é preemptivo puro, optando-se pelo cálculo de um fator de adaptação específico a cada tarefa, levando em consideração sua prioridade ou importância. Ainda, foi escolhido o mínimo múltiplo comum (MMC) do período das tarefas como o período do serviço de adaptação de forma *ad-hoc*, assim como acontece em outros trabalhos na literatura. É uma questão em aberto se esta escolha é a melhor para qualquer aplicação. Experiências para verificar esses aspectos e analisar a escalabilidade da solução, visto que os testes realizados para esse artigo utilizaram apenas duas máquinas, estão entre alguns dos possíveis trabalhos futuros.

Ainda, outras possibilidades de implementação do mecanismo proposto poderiam ser testadas. Ou ainda, uma substituição do serviço de eventos do TAO por outro que privilegiasse especificamente a geração de eventos periódicos e a reconexão dos consumidores poderia ser proposta e validada como uma alternativa para tornar mais rápida a resposta da adaptação.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Mowbray, Thomas J.; Ruh, William A.; *Inside CORBA: Distributed Object Standards and Applications.* USA: Addison-Wesley, 1998.

[2] Orfali, Robert; Harkey, Dan; *Client/Server Programming with JAVA and CORBA,* 2. ed. USA: Wiley Computer Publishing, 1998.

[3] Feng, W.; Syyid, U.; Liu, J. W.-S. *Providing for an Open, Real-Time CORBA.* Disponível por WWW em http://www.researchindex.com (Dez 2000).

[4] O'Ryan, Carlos; Schmidt, Douglas C.; Kuhns, Fred; Spivak, Marina; Parsons, Jeff; Pyarali, Irfan; Levine, David L. *Evaluating Policies and Mechanisms to Support Distributed Real-Time Applications with CORBA.* Disponível por WWW em http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html (Nov. 2000).

[5] Gill, Christopher D.; Levine, David L.; Schmidt, Douglas C. *The Design and Performance of a Real-Time CORBA Scheduling Service.* Disponível por WWW em

http://www.researchindex.com (Dez 2000).
[6] Oliveira, Rômulo S. *Mecanismos de Adaptação para Aplicações Tempo Real na Internet.* Congresso da Sociedade Brasileira de Computação (SEMISH'98) Belo Horizonte, 1998.
[7] Pope, Alan; *The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture.* USA: Addison-Wesley, 1998.
[8] Henning, Michi; Vinoski, Steve; *Advanced CORBA Programming with C++*. USA: Addison-Wesley, 1999.
[9] OMG; *Realtime CORBA.* Alcatel; Hewlett-Packard Company; Lucent Technologies, Inc.; Object-Oriented Concepts, Inc.; Sun Microsystems, Inc.; Tri-Pacif. OMG Document orbos/98-01-08. Disponível por WWW em http://www.cs.wustl.edu/~schmidt/PDF (Out. 2000)
[10] Schmidt, Douglas C.; Levine, David L.; Cleeland, Chris. *Architectures and Patterns for Developing High-performance, Real-time ORB Endsystems.* Setembro, 1998. Disponível por WWW em http://www.cs.wustl.edu/~schmidt/corba-research-realtime.html (Out. 2000)
[11] Schmidt, Douglas C.; Levine, David L.; Mungee, Sumedh. *The Design of the TAO Real-Time Object Request Broker*. Computer Communications Journal, 1997. Disponível por WWW em http://www.researchindex.com (Dez 2000).
[12] Harrison, Timothy H.; Levine, David L.; Schmidt, Douglas C.; *The Design and Performance of a Real-Time CORBA Event Service.* Object-Oriented Programming Systems, Languages and Applications (OOPSLA), Outubro 1997. Disponível por WWW em http://www.researchindex.com (Ago 2001).
[13] Jensen, E. D.; Locke, C. D.; Tokuda, H.; *A Time-Driven Scheduling Model for Real-Time Operating Systems.* Proceedings of the IEEE Real-Time Systems Symposium, pp. 112-122, dez. 1985.
[14] Lu, Chenyang; Stankovic, John A.; Abdelzaher, Tarek F.; Tao, Gang; Son, Sang H.; Marley, Michael; *Performance Specifications and Metrics for Adaptive Real-Time Systems.*21st IEEE Real-Time Systems Symposium, Nov. 2000.
[15] Beccari, G.; Caselli, S.; Reggiani, M.; Zanichelli, F.; *Rate Modulation of Soft Real-Time Tasks in Autonomous Robot Control Systems.* 11th Euromicro Conference on Real-Time Systems, England, Jun. 1999.
[16] Welch, L. R.; Shirazi, B. A.; Ravindran, B.; *Adaptive Resource Management for Scalable, Dependable Real-Time Systems: Middleware Services and Applications to Shipboard Computing Systems.* IEEE Real-time Technology and Applications Symposium, Jun. 1998.
[17] Shin, K.G.; Meissner, C. L.; *Adaptation and Graceful Degradation of Control System Performance by Task Reallocation and Period Adjustment.* 11th EuroMicro Conference on Real-Time Systems, Jun. 1999.
[18] Abdelzaher, T. F.; Atkins, E. M.; Shin, K. G.; *QoS negotiation in real-time systems and its application to automatic flight control.* IEEE Real-Time Technology and Applications Symposium, Jun. 1997.
[19] Abdelzaher, E. M.; Shin, K. G.; *End-host Architecture for QoS-Adaptive Communication.* IEEE Real-Time Technology and Applications Symposium, Jun. 1998.